Using Yocto as a Method to Upstream, Maintain, and Track Patches

Jon Mason

23 August 2019

Problem

- SoCs have multiple open-source components
  - Kernel
  - U-boot
  - ATF
  - Graphics

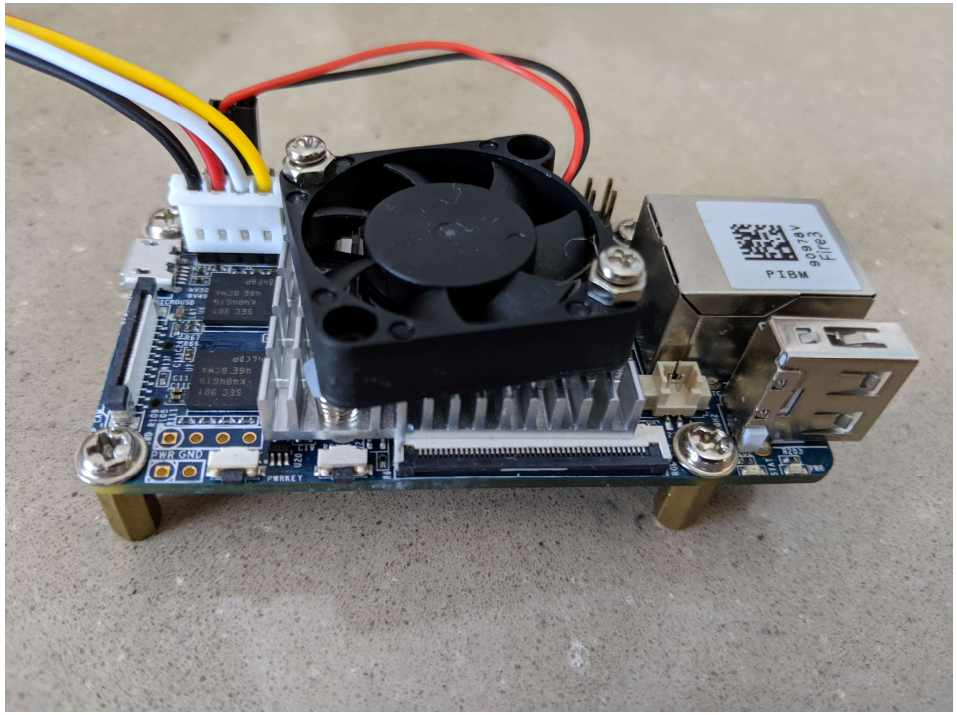
- These need to be upstreamed and maintained
- We need a simple way to manage this

FriendlyArm NanoPi Fire3

- Samsung S5P6818
- Cortex-A53 eight-Core at 1.4GHz

- Vendor supplied “Ubuntu” image
- Kernel (v4.4) and u-boot (2016.01) source available at vendor website

- Used as a guinea pig for this exercise



arm

Groundwork - Conforming existing code to Yocto

The 4 Steps

1. Get it working under Yocto
2. Migrate to generic sources
3. Split into patches
4. Push patches

Get it working under Yocto

1. Commit to code git trees
2. Write recipes
3. Verify bootable image is created

Commit to code git trees

- Some SoC vendors only provide the legally obligated source in tarball
- This can be handled easily by inspect the Makefile for the version, cloning a version of the relevant tree with the relevant version tag, copying over the source, and making an atomic commit on top of the tree. This will show the relevant changes, while being not too messy.

Write recipes

- Recipes for every binary needed for the bootable image are needed, along with a WIC file to specify how an bootable image is created with these binaries
- This is the “hardest part”, as writing recipes from scratch may be necessary, and ugly hacks like copying binaries for sources you do not have access to may be necessary
- Create a unique meta- layer (if necessary)

- Potential huge benefit
  - If this work is being done for a series of boards based on the same SoC, the amount of work to create unique images for each board is almost trivial (after doing the initial board)
  - As of right now, there are 14 boards based on the Yocto layer I created, and it should be trivial to add support for 2 more product lines based on the same SoC

arm

NanoPi Fire 3 recipes

Linux Kernel recipe

```
inherit kernel
require recipes-kernel/linux/linux-yocto.inc
SRC_URI = "git://github.com/jonmason/linux-s5p6818.git;protocol=https;branch=nanopi2-v4.4.y \
           file://s5p4418_defconfig.patch \
           file://s5p6818_defconfig.patch \
          "
SRCREV = "2baec73557c2fe5350160596870f7e1f411c91be"
LINUX_VERSION ?= "4.4.49"
PV = "${LINUX_VERSION}+git${SRCPV}"
LICENSE = "GPLv2 & LGPLv2"
S = "${WORKDIR}/git"
```

U-boot recipe

```
require ${COREBASE}/meta/recipes-bsp/u-boot/u-boot.inc
SUMMARY = "U-Boot bootloader for Samsung/Nexell s5pxx18"
LICENSE = "GPLv2+"
LIC_FILES_CHKSUM = "file://Licenses/README;md5=0507cd7da8e7ad6d6701926ec9b84c95"
# u-boot needs devtree compiler to parse dts files
DEPENDS += "dtc-native bc-native"
SRCREV = "${AUTOREV}"
SRC_URI = "git://github.com/friendlyarm/u-boot.git;protocol=https;branch=nanopi2-v2016.01"
S = "${WORKDIR}/git"
do_configure() {
                mkdir -p ${B}/tools/nexell/nsih/
                cp ${S}/tools/nexell/nsih/nanopi2.txt ${B}/tools/nexell/nsih/
                cp ${S}/tools/nexell/nsih/nanopi3.txt ${B}/tools/nexell/nsih/
                sed -i '/autoconf/d' ${S}/arch/arm/dts/s5p4418.dtsi
}
```

L-loader recipe

```
DESCRIPTION = "l-loader for s5p6818"
LICENSE = "GPLv2+"
inherit deploy

#SRC_URI = "git://git.nexell.co.kr/nexell/secure/l-loader;protocol=git;branch=nexell"
SRC_URI = "git://github.com/friendlyarm/sd-fuse_s5p6818.git;protocol=https;"
SRCREV = "ff5d9d52f6dd8d608a4fc3829976a078d3defe0f"

LIC_FILES_CHKSUM = "file://README.md;md5=0d6c7f42efcf5e2931accdbdf5d2bcfc"

S = "${WORKDIR}/git"

do_deploy() {
                install -m 0644 ${S}/prebuilt/fip-loader.img ${DEPLOYDIR}/
}
```

Verify bootable image is created

- Hook into automated testing framework if possible

Migrate to generic sources

- Rebase the existing code on top of upstream git trees

Updating u-boot recipe (example)

Create a working u-boot version

git clone git://git.denx.de/u-boot.git u-boot
cd u-boot
git remote add friendlyarm https://github.com/friendlyarm/u-boot.git
git remote add github git@github.com:jonmason/u-boot.git
git fetch -tp --all
git push github friendlyarm/nanopi2:nanopi2

arm

Updating u-boot recipe (example)

Change the recipe

```
$ git diff
diff --git a/recipes-bsp/u-boot/u-boot-nexell_2016.01.bb b/recipes-bsp/u-boot/u-boot-nexell_2016.01.bb
index 7381224..d2948c1 100644
--- a/recipes-bsp/u-boot/u-boot-nexell_2016.01.bb
+++ b/recipes-bsp/u-boot/u-boot-nexell_2016.01.bb
@@ -9,7 +9,8 @@ SRCREV = "${AUTOREV}"

-SRC_URI = "git://github.com/friendlyarm/u-boot.git;protocol=https;branch=nanopi2-v2016.01"
+#SRC_URI = "git://github.com/friendlyarm/u-boot.git;protocol=https;branch=nanopi2-v2016.01"
+SRC_URI = "git://github.com/jonmason/u-boot.git;protocol=https;branch=nanopi2"

 S = "${WORKDIR}/git"
```

Updating u-boot recipe (example)

Rebase onto an existing upstream release

```
git checkout -b nanopi2 friendlyarm/nanopi2
git rebase v2016.01
git rebase -i v2016.01
git push -f github nanopi2
```

Build and verify no changes to the recipes are needed

NOTE: not able to cleanly rebase to the latest version do to the amount of changes done between v2016.01 and v2019.01

Updating kernel recipe

```
diff --git a/recipes-kernel/linux/linux-nexell_4.4.bb b/recipes-kernel/linux/linux-nexell_4.4.bb
index 9e30a7a..be67b83 100644
--- a/recipes-kernel/linux/linux-nexell_4.4.bb
+++ b/recipes-kernel/linux/linux-nexell_4.4.bb
@@ -7,11 +7,9 @@ SRC_URI = "git://github.com/jonmason/linux-s5p6818.git;protocol=https;branch=nan
          file://s5p6818_defconfig.patch \
         "

-SRCREV = "2baec73557c2fe5350160596870f7e1f411c91be"
+SRCREV = "${AUTOREV}"

-LINUX_VERSION ?= "4.4.49"
-
-PV = "${LINUX_VERSION}+git${SRCPV}"
+KERNEL_VERSION_SANITY_SKIP = "1"

 LICENSE = "GPLv2 & LGPLv2"
```

Split into patches

Preservation of history is not important, but maintaining copyright is!

Recommend split and order of upstreaming:

1. Base Soc Support (CPU, RAM, UART)
2. Clocks
3. Storage
4. Networking
5. Miscellaneous

arm

Using Yocto as a Method to Upstream, Maintain, and Track Patches

Upstreaming Process

1. Email relevant mailing lists
2. Get feedback
3. Modify patches
4. Repeat

Using Yocto to upstream

How to do the upstreaming with Yocto

- 2 easy ways to create and track
  1. Git tree that is being referenced in Yocto recipe
  2. Patches directory in the meta layer (most likely generated via git tree)

- Rebase source to the latest tag (assuming git tree upstream is tagged)
  - `git rebase -i $TAG`
    - This will either work without problem or be extremely painful, mostly depending on the amount of modifications in the code being touched by people upstream

- Because patches were already split previously and are now on the latest version, all you need to do is `git send-email`

Using Yocto to Track patches

Update the upstream recipe to track the latest version. As each patch is pulled in, git will automatically skip over it when rebasing. Thus the delta between the latest tag and the internal HEAD will be the queue of outstanding patches.

- If using the git method of upstreaming the patches, rebase the git tree on the newest upstream tag and modify the SRCREV hash in the recipe (or better yet, set to AUTOREV)
- If using the directory of patches, you might need to regenerate them and recommit to the meta-layer git tree when the changes no longer merge cleanly when the recipe is building.

Using Yocto to maintain

- Updating recipes (or using the upstream recipes) to track the latest versions
- Use the “tracking” method to verify patches are accepted upstream

Push your layer to the OE layer index

Finally…

- Recipes and relevant code can be found at https://github.com/jonmason/meta-samsung

arm

Thank You
Danke
Merci
谢谢
ありがとう
Gracias
Kiitos
감사합니다
धन्यवाद
شكرًا
תודה